Document Identification: idlosdoc
Document Revision: 1.4

IDLos doc Tags

DOCUMENT CHANGE RECORD

Revision	Date	Author(s)	Description
1.0	June 30, 2004	A. Chou	• Extract Appendix A from doc.tech.002.doc.
1.1	July 12, 2004	A. Chou	• Removed "2.2.14 XHTML Tags" section.
1.2	July 19, 2004	A. Chou	• Update comments as built.
1.3	September 7, 2004	A. Chou	• Update to as-built. • Add an example.
1.4	September 13, 2004	A. Chou	• Remove the need of '*' to do the left alignment.

REVIEWER LIST

Name	Title	Location
D. Sundstrom	Components Group Manager	Austin, TX
D. Sifter	CTO	San Rafael
E. Bulanin	Quality Assurance	San Rafael
A. Fuchs	Documentation Technical Lead	San Rafael

TABLE OF CONTENTS

1 INTRODUCTION 1

1.1 MOTIVATION 1
1.2 OBJECTIVES 1
1.3 AUDIENCE 1

2 IDLOS DOC TAGS 2

2.1 MODEL DOCUMENTATION STRUCTURED COMMENT SYNTAX 2
2.2 SEMANTIC TAG SYNTAX 2
2.2.1 TAG TABLE 3
2.2.2 @ABSTRACT 3
2.2.3 @COPYRIGHT 3
2.2.4 @DEPRECATED 4
2.2.5 @DISCUSSION 4
2.2.6 @EXAMPLE 4
2.2.7 @EXCEPTION 4
2.2.8 @PARAMETER 5
2.2.9 @PREFORMATTED 6
2.2.10 @PRIVATE 6
2.2.11 @RESULT 6
2.2.12 @SEE 6
2.2.13 @WARNINGS 7
2.2.14 DEPRECATED IDLOSDOC TAGS 7
2.2.15 WHITE SPACE CONTROL 7
2.2.16 TAG POSITIONING 7
2.2.17 SPECIAL TAG CHARACTER 7
2.2.18 SPECIAL HTML CHARACTERS 7
2.2.19 XHTML TAGS SUPPORT 8
2.2.20 LEFT ALIGNMENT 8

3 EXAMPLE 9

3.1 EXAMPLE 9

1 INTRODUCTION

1.1 Motivation

To help developers to use the supported IDLos doc tags in the Kabira documentation system.

1.2 Objectives

This document specifies the IDLos doc tags supported by the Kabira documentation system and their usages.

1.3 Audience

Everyone who will use it.

2 IDLOS DOC TAGS

2.1 Model Documentation Structured Comment Syntax

All structured comment blocks must start with the three-character anchor "/**" and end with the two-character sequence "*/". The text enclosed by these two anchors is considered to be embedded documentation. If an element has no structured comment, it will not be documented.

There are three kinds of tagged comments. One designates the semantic content of the comment, and is specified using an "@" symbol. The others allow the developer to specify structural and content for model documentation, and they conform to XML syntax.

2.2 Semantic Tag Syntax

This section describes semantic tags. Semantically tagged comments are embedded in structured comment blocks, and have the following format:

```
/**
       @<tag name 1> [< name 1>] [<Description 1>]
       ...
       @<tag name n> [< name n>] [<Description n>]
*/

<code block associated with the structured comment>
```

The "@<...>" constructs are the actual tags used to define different kinds of item descriptions.

Each tag must begin on its own line and all leading white spaces are ignored. This tagged paragraph also includes any following lines up to, but not including, either the first line of the next tag or the end of the structured comment block as indicated by the "*/" anchor. The definition of the tagged item follows the "*/" anchor on the next line.

2.2.1 Tag Table

The following table lists the structured comment tags supported by the document system.

Tag	Description	Constraints
@abstract	short description of the element	One per element
@copyright	copyright notice	One per package
@deprecated	marks the element as deprecated	One per element
@discussion	longer descriptive text	One per element
@example	code example. The @example tags are indexed as examples in the documentation.	None
@exception	describes an exception raised by the operation	Required; one per raised exception in operation
@parameter	operation parameter	Required; one per parameter in operation
@preformatted	preformatted text. All white space and formatting is preserved.	None
@private	do not generate IDLos documentation output for this element, or any elements contained within this element.	One per element
@result	operation return value	Required if operation has structured comment
@see	hyperlink that uses simplified code to identify the target within the document system.	None
@warnings	textual warnings	None

Table 2: Semantic Tags

2.2.2 @abstract

The @abstract tag is an optional tag that precedes a documentable model element. It is used to provide a brief one sentence summary of the element. If no @abstract is provided, the first sentence of @discussion will be used. The usage:

```
/**
  @abstract IDLos static definitions for HTTP client.
 */
```

2.2.3 @copyright

The @copyright tag is used to insert a copyright statement for package in the model documentation. If none is provided, the following line will be used:

```
Copyright © Kabira Technologies, Inc. All rights reserved.
```

The usage:

```
/**
  ...
  @copyright 2003 by Kabira Technologies, Inc. Confidential Property
  ...
*/
```

2.2.4 @deprecated

The @deprecated tag is used to document an element that should no longer be used (though it may still work). The usage:

```
/**
@deprecated This function is replaced by this operation.
*/
```

2.2.5 @discussion

The @discussion tag is an optional tag that is used to create general documentation for a model element. There can be only one @discussion tag per model element.

The usage:

```
/**
@discussion List of supported HTTP methods.
*/
enum MethodType
{
    /** HTTP OPTIONS method. */
    OptionsMethod,
    /** HTTP GET method. */
    GetMethod,
    /** HTTP HEAD method. */
    HeadMethod,
    /** HTTP POST method. */
    PostMethod,
    /** HTTP PUT method. */
    PutMethod,
    /** HTTP DELETE method. */
    DeleteMethod,
    /** HTTP TRACE method. */
    TraceMethod
};
```

2.2.6 @example

The @example tag can be used to add code snippets to model element documentation. The white space and line breaks are preserved in the resulting output.

The usage:

```
/**
  @example
  entity E
  {
     attribute TimerHandle timerHandle;
  };

 */
```

2.2.7 @exception

The @exception tag is used to document an exception raised by an operation.

Note: @exception is not used for the exception repository element but the operator element.

The usage:

```
/**
 @discussion Your CPU has melted.
 */
exception Melt
{
  /** Read about your melted CPU */
  string message;
};
/**
 @discussion Your water has boiled.
 */
exception Boil
{
  /** Put your tea bag in the water */
  string message;
};
/**
*
@discussion This operation does something that results in a great deal
of heat.
@exception Melt This is raised when the operation causes the CPU
to melt.
@exception Boil This is raised when the operation causes your tea
water to boil.
*/
void doSomething() raises (Melt, Boil);
```

2.2.8 @parameter

The @parameter tag documents a parameter in an operation. It is usually accompanied by a @result tag, which documents the return type for the operation.

Note: Lacking either `<name>` or `<description>` is a syntax error and no documentation will be generated.

The usage:

```
/**
    @abstract    receive a line with timeout.
    @discussion
        Called to get a line (i.e. sequence of bytes
        delimited by CRLF) from the HTTP server.
    @parameter       timeout      time out value in seconds.
    @parameter       responseLine response line.
    @result
        can be one of
            OperationSucceeded,
            Disconnected,
            OperationTimedOut,
            OperationFailed
*/
Status recvLineTimeout(in  long   timeout,
                       out string responseLine);
```

2.2.9 @preformatted

You can use the @preformatted tag to create preformatted text for your documentation. The whitespace and line breaks are preserved in the resulting output.

The usage:

```
/**
 ...
 @preformatted
   This is pre-formatted text:
     Here's one line.
        This one's indented further.
 ...
*/
```

2.2.10 @private

The @private tag instructs the documentation system not to generate any output. This disabling of output applies to the element, and any elements contained by the element.

The usage:

```
/** @private */
interface interfaceNoDoc
{
       attribute char c;
};
```

2.2.11 @result

The @result tag is used to document the result returned by an operation. For more information see @parameter.

2.2.12 @see

The @see tag allows you to create the following 3 types of reference hyperlink to other documentation:

- the full URL

  ```
  /**
   @see http://www.kabira.com
   */
  ```

- the other repository element in the SAME page

  i.e. In package content, `content::fields::Resolution` can refer to `content::ReserveResolution` as follows:

  ```
  /**
   @see ReserveResolution
   */
  ```

- the relative path or file names from `$SW_HOME`

  ```
  /**
   @see distrib/kabira/kts/config/samples/
   */
  ```

or

```
/**
 @see distrib/kabira/kts/config/samples/sampleuserdata.xsd
 */
```

2.2.13 @warnings

The @warnings tag is an optional tag that can be used to provide a description of any appropriate warnings. The usage:

```
/**
 @warnings This is a test for the warnings heading.
*/
```

2.2.14 Deprecated IDLosdoc Tags

The following deprecated tags **will fail** at component build time if they are present:

- `@package`
- `@also`
- `@const`
- `@enum`
- `@param`
- `@field`
- `@struct`
- `@union`
- `@operation`
- `@interface`
- `@attribute`
- `@typedef`
- `@relationship`
- `@role`
- `@url`

2.2.15 White space Control

In structured comments, the system will convert two newlines (or an empty line) into a paragraph break

2.2.16 Tag Positioning

Semantic tags must be placed inside of structured comments immediately before the model element they describe, with the exception of `@parameter` , `@exception,` and `@result` tags, which are grouped in the structured comment block preceding the operation to which the parameters belong.

2.2.17 Special Tag Character

To display @ in the documentation, use @@ to avoid the document system to treat it as an invalid tag.

2.2.18 Special HTML Characters

To display < and > in the documentation, one should use their HTML code as `<` and `>` accordingly to avoid them getting interpreted as HTML tags.

2.2.19 XHTML Tags Support

The documentation system supports well-formed HTML tags. (XHTML implies well-formed HTML.) The definition of well-form is to have a pair of HTML tags such as

```
<pre></pre>
```

or

```
<p/>
```

2.2.20 Left Alignment

The documentation system will find the line with the fewest white spaces in the IDLos doc block and remove the number of white spaces from each line. For example,

```
package test
{
        /**
        @preformatted
        if a in plan
                put a tab in the line
          else if b in plan
                        put 2 tabs in the line
        done
        */
        interface ifc
        :
```

This will generate the following text in the browser:

Preformatted:

```
if a in plan
        put a tab in the line
  else if b in plan
                put 2 tabs in the line
done
```

3 EXAMPLE

The default behavior during component building:

- *.sdl files generate documentation.
- *.soc files do not generate documentation.

To force different behavior, the user can set the registry key “documentation” as follows:

```
component test
{
        source test.sdl;
        source test.soc;
        source test2.soc
        {
                document = true;
        };
        source test2.sdl
       {
                document = false;
        };
        package test;
};
```

3.1 Example

```
/**
@abstract       Package test displays how one uses tags.
@discussion     We'll demonstrate how one can use tags like @@abstract,
                @@discussion, and so on.

One can also use the HTML tags as follows:
        <ol>
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
        </ol>

                This is a complete set of tags supported by package.
                The &lt;pre&gt; HTML tag is supported now.
                <pre>
        NOTE:
        -----
        It reserves all the white space at the beginning of the line.
                </pre>
@copyright      Copyright of Kabira Technologies, Inc.
@warnings       This is the 1st warnings.
@example
You can do something like the followings:

    test::negativeFoo = 12;
    test:typedefString = "testing";

@preformatted
        The first @@preformatted tag.
        It preserves all the white space as you can see from
```

```
              the generated page.
@warnings       This is the 2nd warnings.
@warnings       This is the 3rd warnings.
@see            http://www.kabira.com
@see            interfaceBar
@see            distrib/kabira/kts/config/samples
@see            distrib/kabira/kts/config/samples/sampleuserdata.xsd
@example
A 2nd example.
@preformatted
                The 2nd preformatted tag to
        demonstrate that you can have more than
  one preformatted tag.
*/

package test
{
        /**
         * @discussion A const string named constString.
         */
        const string constString = "<STRING>";

        /**
                @abstract A typedef string named typedefString.
        */
        typedef string typedefString;

        /**
        A sequence typdef of long named typedefSeqLong.
        */
        typedef sequence<long> typedefSeqLong;

        /** @abstract A native SW_INT32 named nativeFoo. */
        native SW_INT32 *nativeFoo;

        entity bar;
        interface interfaceFoo;

        /** An exception named ExceptionFooError */
        exception ExceptionFooError
        {
                /** An message for exception ExceptionFooError. */
                string messageFoo;
        };

        /**
         @discussion
              A structure is a very useful thing.
         */
        struct FooCase
        {
                /** A string label */
                string    label;
                /** An object element */
                Object    element;
        };

        /**
```

```
        @abstract       An enum named Color.
        @discussion     This enum is used as the input selection
                        for unionColor.
*/
enum Color
{
        /** The color Red.*/
        Red,
        /** The color White.*/
        White,
        Blue
};

/**
 @abstract     A union named unionColor.
 @discussion  This switch is based on enum Color.
 @warnings     Only valid colors are Red, White, and Blue.
*/
union unionColor switch(Color)
{
        /**
         @abstract      Color Red is a short red.
        */
        case Red:
                short red;
        /**
         @discussion    Color White is a long white.
        */
       case White:
                long white;
        /** Color Blue is a string blue.*/
        case Blue:
                string blue;
};

/**
 @abstract    A module which is like a package except no
              copyright.
 @discussion  This is module moduleTest in package test.
 @deprecated   This should not be used any more.
 */
module moduleA
{
        /** a const in moduleA */
        const char constA = 'a';

        /** @private */
        const short constModuleAPrivate = 42;

        /** moduleB to test the nested module */
        module moduleB
        {
                /**
                @abstract An entity named entTest inside
                            of module.
                */
                entity entTest
                {
```

```
                        /** an attribute long named attLong */
                        attribute long attLong;
                };
        };
};

entity foo
{
        attribute boolean groupBroadcast;
        attribute string fooPrivateStringAttr;
        void setGroupBroadcast(in boolean b);
};

entity bar
{
        attribute long a;
        key Primary { a };
        long setOperationWithException(in  string s,
                out long times)
               raises(ExceptionFooError);
};

/**
 @discussion An interface named interfaceFoo. */
interface interfaceFoo
{
        /** const in interfaceFoo */
        const unsigned long constIfc = '\0';

        /**
         @abstract Set the groupBroadcast.
         @parameter b   Turn on/off the groupBroadcast.
        */
        void setGroupBroadcast(in boolean b);
};
expose entity foo with interface interfaceFoo;

/** @abstract An interface named interfaceBar */
interface interfaceBar
{
        /**
        @attribute a a legacy tag @@attribute a
        @abstract interfaceBar's attribute a */
        attribute long a;

        /**
         @discussion a primary key of attribute a.
         */
        key Primary { a };

        /**
         @discussion  The discussion for this operation.
         @parameter   s             a string
         @parameter   times   number of times to have bar
         @result      number of foo exceptions
         @exception   ExceptionFooError an exception food error
         @example
         //
```

```
                        // Action language
                        //
                        declare string  s;
                        declare long    times;
                        declare long    count;
                        declare ExceptionFooError efe;

                        count = setOperationWithException(s, times)
                                raises(efe);
                       */
                       long setOperationWithException(in string s, out long
times)
                               raises(ExceptionFooError);
               };
               expose entity bar with interface interfaceBar;

               /**
                 @abstract This is relationship FooBarRelationship
                 @example
                      relationship FooBarRelationship
                      {
                              role foo FooToBar 0..* bar;
                              role bar BarToFoo 1..1 foo;
                      };
                */
               relationship FooBarRelationship
               {
                       /** @abstract a role named FooToBar
                       @discussion role FooToBar maps from foo to bar
                       */
                       role foo FooToBar 0..* bar;

                       /** @abstract a role named BarToFoo
                       @discussion role BarToFoo maps from bar to foo
                       */
                       role bar BarToFoo 1..1 foo;
               };

               /** @private */
               const long JohnSilver = 8;

               /** @private */
               interface MYOB
               {
                       attribute char c;
               };
               entity MYOBImpl
               {
                       attribute char c;
               };
               expose entity MYOBImpl with interface MYOB;

               action test::foo::setGroupBroadcast
               {`
                       self.groupBroadcast = b;
               `};

               action test::bar::setOperationWithException
```

```
                {`
                        return 1234;
                `};
};
```